ANALOG DEVICES

# AN-415 APPLICATION NOTE

ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

# ADSP-2181 IDMA Interface to Motorola MC68300 Family of Microprocessors

## INTRODUCTION

The speed and mathematical capabilities of DSP processors, combined with their low cost and expanded integration, make them a natural choice for use as signal co-processors in embedded environments. When paired with a host microprocessor, a DSP processor allows for a very powerful and flexible system at a reasonable price.

The ADSP-2181 is an ideal candidate for use in a co-processing system. The 32K words of on-chip SRAM and extensive DMA and peripheral interface features allow the ADSP-2181 to function with minimal external support circuitry.

In order to realize the highest possible performance in a co-processor system, efficient host-DSP communication is vital. The popular Motorola M68300 Family of microcontrollers provides a powerful and flexible bus interface that is easily adaptable to a co-processing system.

This application note describes an example hardware and software interface between the Internal DMA (IDMA) Port of the ADSP-2181 and the Motorola M68300 Family of microcontrollers. As each specific system design has its own requirements and challenges, this application note does not presume to provide the only possible solution. Rather it is meant to provide the system designer a flexible framework of ideas that can be tailored to meet individual system requirements.

### IDMA Operation

External devices can gain access of the ADSP-2181's internal memory through the DSP's IDMA Port. Host processors accessing the ADSP-2181 through IDMA can treat the DSP as a memory-mapped slave peripheral, and can have access to all of the DSP's internal Data Memory (DM) and Program Memory (PM).

The ADSP-2181's IDMA Port consists of a 16-bit multiplexed address/data bus (IAD16:0), a select line ($\overline{IS}$), address latch enable (ALE), read ($\overline{IRD}$), write ($\overline{IWR}$), and acknowledge ($\overline{IACK}$) signals. The host processor is responsible for initiating all data transfers. A typical transfer sequence is shown in Figure 1.

The DSP memory address is loaded into the IDMA Address register (IDMAA) shown in Figure 2. This register contains the 14-bit internal memory address, along with a bit to specify the type of transfer: 24-bit Program Memory opcodes, or 16-bit Data Memory data. The IDMAA register can be initialized by either the DSP or by a host processor. The host can initialize this register by performing an address latch cycle. An address latch cycle is defined by the host asserting the ALE signal, and then transferring a 15-bit (14 address bits plus 1 destination memory type bit) value on the IAD pins. To streamline the

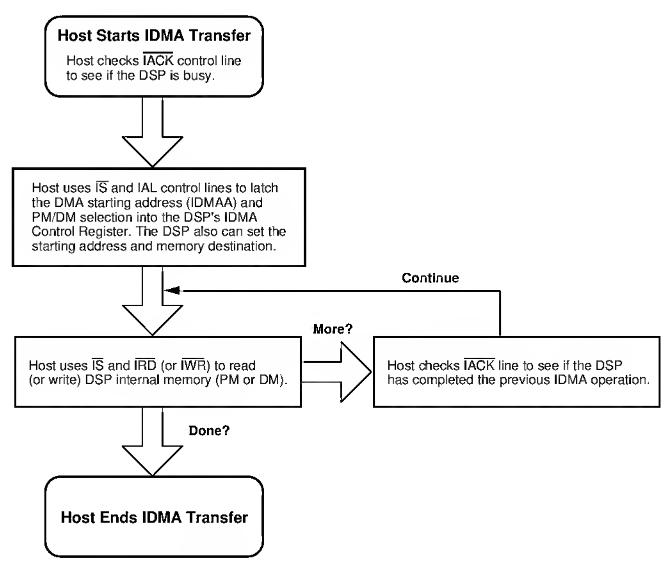



*Figure 1. IDMA Transfer Sequence*

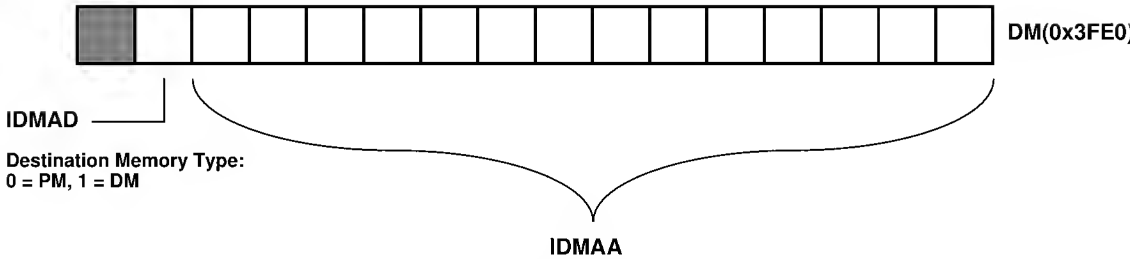



*Figure 2. IDMA Control Register*

transfer of large segments of opcodes or data, an Address Latch Cycle does not need to be performed for each IDMA access. Instead, once latched, the address is automatically incremented after every IDMA word transfer.

As the IDMA Port has a 16-bit bus, 24-bit transfers require two host accesses. The first access transfers the most significant 16 bits, the second access transfers the least significant 8 bits, right justified, with a zero-filled upper byte. IDMA address increments occur after the entire 24-bit word has been transferred.

For more information about the IDMA Port see the *ADSP-21xx Family User's Manual* (Third Edition) and the ADSP-2181 Data Sheet.

## INTERFACE HARDWARE DESIGN

The IDMA Port of the ADSP-2181 is mapped into two locations in the microcontroller's external memory space. One location is used by the microcontroller to set the DSP memory address it wishes to access, and the other location is used when transferring data and instruction information.

### MC6833x Overview

The Motorola MC6833x Family of microprocessors use a System Integration Module (SIM) to communicate to parallel peripherals. The SIM incorporates separate address and data busses, along with multiple memory select lines and strobe lines. The SIM is common (with minor changes) to all MC6833x processors, and material presented in this application note should apply to all processors in the family.

### Schematic Explanation

Minimal logic is required to connect the external bus of the MC6833x to the IDMA Port. All logic necessary for this interface was programmed into a single GAL20V8B programmable logic device. Schematic drawings at both the chip and gate level are provided at the end of this application note, along with the appropriate logic equations.

The 16 data lines from the MC6833x are connected directly to the ADSP-2181's IAD pins. The 6833x will use this bus to transmit the DSP memory address, as well as transfer data to and from the DSP processor.

The $\overline{\text{IACK}}$ signal from the DSP is routed to both the $\overline{\text{DSACK1}}$ pin and a programmable flag pin on the MC6833x. The $\overline{\text{DSACK1}}$ pin signals the end of a memory transfer cycle for the MC6833x, while the programmable flag pin is used by the MC6833x to check $\overline{\text{IACK}}$ status prior to initiating a transfer. The microcontroller downloader code, presented in the Code Listing section of this application note, checks for a low level of the flag prior to any transfer.

The microcontroller's address pin A1 is connected directly to the ALE pin of the IDMA port. To begin a transfer, the microcontroller must first initialize the DSP's IDMAA register through an Address Latch cycle. This is accomplished by writing the DSP memory address that the microcontroller wants to access to address 0xbbb2 in the microcontroller's memory space. The setting of the base address is described in the next paragraph. Address pin A1 was used because it is the lease significant address pin used by the microcontroller during 16-bit word transfers.

Assigning the base address that the ADSP-2181 IDMA port resides at is accomplished through the use of the MC68333's address lines A12 and A13, in conjunction with the microcontroller's $\overline{\text{DS}}$ signal. These signals are combined such that the IDMA Port's $\overline{\text{IS}}$ signal is asserted (low) when $\overline{\text{DS}}$ is asserted (low), A12 is low, and A13 is high. With this combination, the IDMA Port can be accessed in the microcontroller's memory space at addresses 0x2xxx, 0x6xxx, 0xaxxx, and so on. In this application example we use address 0x2000 for data transfers and 0x2002 for IDMA address transfers. Tighter assignment of addresses can be accomplished through the use of additional address lines in the $\overline{\text{IS}}$ logic.

The final IDMA control lines that need to be driven by the 68333 are $\overline{\text{IRD}}$ (IDMA Read) and $\overline{\text{IWR}}$ (IDMA Write). Since the microcontroller only has a single, multiplexed R/$\overline{\text{W}}$ (Read/$\overline{\text{Write}}$) line, the R/$\overline{\text{W}}$ line is inverted and then routed to $\overline{\text{IRD}}$ to generate the IDMA read signal. The IDMA Write signal, $\overline{\text{IWR}}$, is the OR'ed combination of the microcontroller's R/$\overline{\text{W}}$ line, and address line A2. This logic is necessary to insure that $\overline{\text{IWR}}$ stays high during an IDMA Address Latch cycle.

The 74HCT244 in the schematic is a tristatable bus buffer active during microcontroller RESET. It is used to configure bus interface functionality on the 68F333.

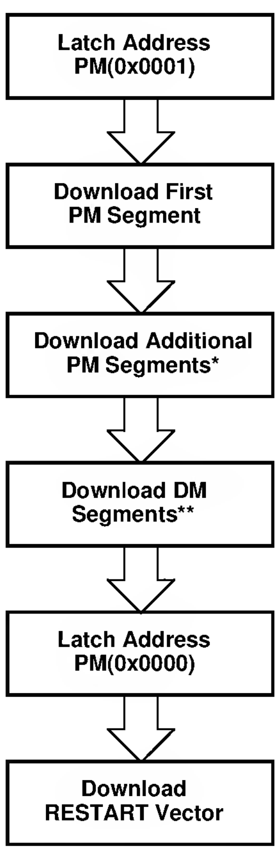
## SYSTEM DESIGN ISSUES

The physical hardware interface between the microcontroller and DSP is just the enabling step in a DSP-based co-processing system. System start-up and host-DSP

communication issues must be planned for ahead of time and adequate provisions for these issues should be included into both the microcontroller's and the DSP's firmware.

**Booting The DSP**

The IDMA Port on the DSP can be used to boot load the DSP on power up. This eliminates the need for a separate EPROM for the DSP. On the ADSP-2181, booting is controlled through the use of the MMAP and BMODE pins. Booting through the IDMA port is enabled by holding the MMAP pin low, and the BMODE pin high. With this signal combination, on RESET, the DSP does not activate its external address bus to access an EPROM. Instead, the DSP expects a host to begin IDMA transfers to fill its internal Data and Program memories. This process consists of the host performing standard IDMA instruction and data transfers. Booting is terminated when the DSP restart vector at DSP Address PM(0x0000) is written. An efficient boot loading sequence would consist of the host filling the DSP's internal Program Memory starting at location PM(0x0001), and using the automatic address increment feature on the IDMA port to speed the transfer of code block in ascending address order. The host can then initialize data memory. When all initialization is complete, the host should then initialize the DSP's restart vector and DSP program execution will commence. This process is shown in Figure 3.




***Each segment download requires its own address latch cycle.**
****DM segments can be downloaded first, or intermixed with PM segments.**

*Figure 3. IDMA Booting Process*

**Generating "Boot" Code**

The ADSP-21xx Family operates on 24-bit instruction opcodes. The IDMA port can only accept 16-bit values. To transfer instruction opcodes through the IDMA port, the most significant 16 bits transferred first, followed by the least significant 8 bits, right justified.

DSP Executable files are produced by the ADSP-21xx Family Linker. The Linker takes object files generated by the Assembler and C Compiler, places them within the memory architecture defined by the system architecture file, and generates a DSP executable (.EXE). The ADSP-2100-Family .EXE format separates Program Memory and Data Memory segments on a module-by-module basis. The executable format has the following elements:

```
@PA          <—— Start of PM RAM Module
0000         <—— Starting address
123456       <—— First Opcode
789abc       <—— Second Opcode
def012       <—— Third Opcode
  :                :
#12345678    <—— End-of-module specifier
@DA          <—— Start of DM RAM segment
0000         <—— Starting address
1234         <—— First data word
5679         <—— Second data word
  :                :
  :                :
#12345678    <—— End-of-module specifier
```

A more detailed description is available in Appendix B of the *ADSP-2100 Family Assembler Tools & Simulator Manual.*

Since the executable file contains 24-bit DSP opcodes, the file needs to be re-formatted to adhere to the IDMA port's 16-bit data requirement. This re-formatting can be accomplished through the use of a C program run on the PC that examines the location and address information contained in the .EXE file. This program can create a data file that can be included as a data buffer into a 6833x assembly file. An example C program for format conversion, IDMABOOT.C, is provided in the Code Listing section of this application note.

Data segments and program segments can be intermixed in a .EXE file. The conversion program must therefore check the start of every module to determine the appropriate memory destination.

***PM Opcode Issues***

ADSP-2100 Family opcodes are 24 bits wide, and the IDMA Port of the ADSP-2181 is 16 bits wide. As described earlier, in order to transmit 24-bit values to PM, the most significant 16 bits get transferred first, followed by the least significant 8 bits, right justified with leading zeros. The conversion program must recognize the @PA symbol, and format the opcodes accordingly. For PM transfers, the IDMAD (IDMA Destination) bit in the IDMA control register is set to 0, so the linker generated address can be retained. The download scheme is described in detail in the next section.

### *DM Address Issues*

To download 16-bit data to DM, the data itself does not need to be altered. However, the linker-generated DM segment address needs to have bit 14 set, as that signifies a DM transfer in the IDMA control register. This alteration is performed in our conversion program by ORing the DM segment address read from the .EXE file with the value 0x4000. Using this bit as a 15th address bit, the MC6833x can access the two separate DSP memory spaces as a single unified address space.

In this application, we add one more piece of information to the re-formatted executable file, a word count. This allows the microcontroller to know how many words of data it needs to transfer for each module or memory segment.

### Host Code Generation - Downloading Issues

In order to utilize the data file produced by the IDMABOOT.C program, the microcontroller needs to be programmed to understand the given format.

The IDMABOOT.C program produces a data file that can be initialized somewhere in the microcontroller's memory space. The first element read from the data file is the number of 16-bit words to be transferred to the DSP (remember that each 24-bit PM opcode counts as two 16-bit words). This value is placed in a data register and can be used as a loop counter to control the download function. The next value in the data file is the DSP starting address of that code or data segment. This is treated as a single 15-bit value as described above. The next values are the data or instruction values that need to be transferred. When the microcontroller has transferred the proper number of items (as determined by the count), it gets the next count value from the buffer, the next DSP address, and so on. The download process stops when the microcontroller encounters a count value of 0xffff. This process is diagrammed in Figure 4. MC68F333 assembly code to implement this download process is presented in the Code Listing section, download.asm.

### Host-DSP Message Transfers

In addition to boot-loading the DSP, many systems require continuous interaction between a host microcontroller and the DSP computation engine. The IDMA port of the ADSP-2181 was designed such that there does not need to be any DSP core involvement with host microcontroller transfers. The host processor is expected to manage the data flow to and from the DSP.

No DSP interrupts are generated during IDMA accesses, and IDMA transfers occur asynchronously to DSP operation. The system designer must therefore allocate DSP internal memory resources and arbitrate host accesses such that there is no conflict between host access and DSP access of DSP internal memory resources.

For data transfers, one could allocate an area of internal memory for "messages", and constrain the host to access this area only. For code transfers other than booting, a software flag set in this "message" area could be used to signal the host that the DSP is available for transfer.

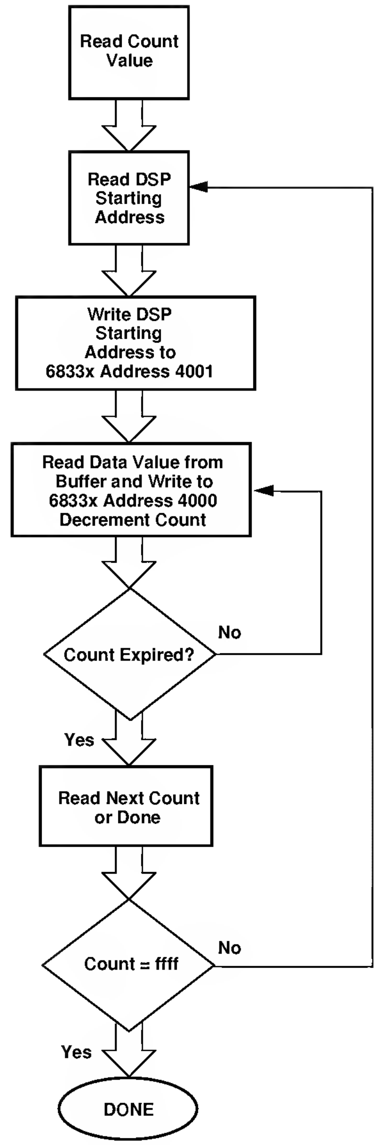



*Figure 4. MC6833x Download Flow*

## TOPICS FOR FURTHER DISCUSSION

### Hardware Signaling

In many instances, it may be desirable for the host and DSP processors to have additional avenues of communication. The host can use one of its programmable flags as an output attached to a hardware interrupt on the DSP. With this method, the host can alert the DSP prior to a transfer, or inform the DSP that a transfer has been completed. This can be especially useful because there is no interrupt associated with IDMA operation on the ADSP-2181. The DSP can likewise use a programmable flag as an output to signal the host if there is new data for the host to use, or if new code is required for download.

### Multiple DSP Processors

In this application note, we focused on connecting a single ADSP-2181 to a microprocessor. This scheme can be expanded to multiple DSP processors without too much trouble.

In a multiple DSP system, all IDMA lines except $\overline{IS}$ and $\overline{IACK}$ can be bussed together. Multiple IS lines are needed to select each individual DSP processor, and multiple $\overline{IACK}$ lines are needed to monitor the activity on each individual DSP processor. Each DSP processor needs two microcontroller memory space addresses assigned to it, and that address assignment used to assert the appropriate $\overline{IS}$ signal. Each DSP processor can then be accessed individually.

**ABOUT THIS APPLICATION NOTE**

The Motorola MC6833x code for this application note was developed using the DEV333 Development Kit from P&E Microcomputer Systems, Inc., Woburn, MA. The hardware interface was verified through the use of an MC68F333-based DEV333KIT evaluation board and an ADSP-2181-based EZ-KIT Lite (ADDS-21xx-EZLITE) from Analog Devices.

**REFERENCES**


**ADSP-21xx Family User's Manual, Third Edition (9/95)**
Analog Devices, Inc.
**ADSP-2100 Family Assembler Tools & Simulator Manual**
Analog Devices, Inc.
**ADSP-2100 Family EZ-KIT Lite Reference Manual**
Analog Devices, Inc.
**M68300 Family CPU32 Reference Manual**
Motorola, Inc. (reference number CPU32RM/AD)
**Modular Microcontroller Family SIM Reference Manual**
Motorola, Inc. (reference number SIMRM/AD)
**MC68F333 User's Manual**
Motorola, Inc. (reference number MC68F333UM/AD)
**68F333 Development Kit User's Manual, Revision 1.00**
P&E Microcomputer Systems, Inc.


**SOURCES**

Analog Devices
Computer Products Division
1 Technology Way, P.O. Box 9106
Norwood, MA 02062-9106
1-800-ANALOGD (literature and technical support)
(617) 461-4258 (BBS)
ftp.analog.com (ftp site)
http://www.analog.com (World Wide Web)

Motorola Literature Distribution
P.O. Box 20912
Phoenix, AZ 85036
1-800-441-2447

P&E Microcomputer Systems, Inc.
P.O. Box 2044
Woburn, MA 01888-0044
(617) 353-9206 (voice)
(617) 353-9205 (fax)
(617) 353-9204 (BBS)

**CODE LISTINGS**

```
; download.asm
;
; This code runs on an MC6833x processor and is used to download
; code and data segments to an ADSP-2181 IDMA port interface
;
SCDR    EQU   $fffc0e          ;SCI Data Register
SCCR0   EQU   $fffc08          ;SCI Control Register 0
SCCR1   EQU   $fffc0a          ;SCI Control Register 1
QMCR    EQU   $fffc00          ;QSM Configuration Register
SCSR    EQU   $fffc0c          ;SCI Status Register
SRAMBAH EQU   $fffb44          ;SRAM Base Address Register High Word
SRAMMCR EQU   $fffb40          ;SRAM Module Configuration Register
FYPCR   EQU   $fffa21          ;SCIM System Protection Control Register
SIMMCR  EQU   $fffa00          ;SCIM Configuration Register
CSPAR0  EQU   $fffa44          ;Chip Select Pin Assignment Register 0
CSPAR1  EQU   $fffa46          ;Chip Select Pin Assignment Register 1
CSBAR0  EQU   $fffa4c          ;Chip Select Base Register 0
CSOR0   EQU   $fffa4e          ;Chip Select Option Register 0
PORTF0  EQU   $fffa18          ;Port F Data Register
;    6833x MEMORY MAP:
;            $000000-$0003FF   Interrupt Vector Table    {TRAM}
;            $000400-$000DFF   Code Space                {TRAM}
;            $010000-$0101FF   Variables (left blank)    {SRAM}
;            $0101FF-Downward  Stack Space               {SRAM}
;
```

```
;*************
; Variables
; DSP Code and Data will be placed here
;*************
   org $010000

; Opcode and data information for DSP download should be included here

   org $000400

;*************************************************************************
;Init: Beginning of the CODE segment
;*************************************************************************

Init:
   move.b #$0,(FYPCR).L        ; Turn off watchdog timer
   move.l #$101FE,a7           ; Stack at location $101FE
   move.w #$0001,(SRAMBAH).L   ; Move SRAM to $10000
   move.w #$0000,(SRAMMCR).L   ; Turn on SRAM (Variables/Stack)
   ;move.w #$0040,(SIMMCR).L   ; Enable User Mode
   move.w #$3FFF,(CSPAR0).L    ; Enable Chip Selects 0-5
   move.w #$03FF,(CSPAR1).L    ; Enable Chip Selects 6-10
   move.w #$0000,(CSBAR0).L    ; Use Chip Select 0
   move.w #$3822,(CSOR0).L     ; Assert Chip Select 0

top:
   move.w (PORTF0).l,d1        ; Check PF1 to see if IACK low before
   and.w #$0002,d1             ; proceeding
   bne top
   move.l #$002002,a4          ; initialize a4 with Address Latch address
   move.l #$002000,a3          ; initialize a3 with data port address
   move.l #$010000,a2          ; initialize a2 to start of DSP code/data
   move.w (a2)+,d2             ; load count value into d2

tx_rx_loop:
   move.w (PORTF0).l,d1        ; check PF1 to see if IACK low
   and.w #$0002,d1
   bne tx_rx_loop
   move.w (a2)+,(a4)           ; write starting address to IDMAA
tx_data:
   move.w (a2)+,(a3)           ; transfer next instruction
wait_data:
   move.w (PORTF0).l,d1        ; check PF1 to see if IACK low
   and.w #$0002,d1
   bne wait_data
   dbf d2,tx_data              ; decrement count to see if at end of module

   move (a2),d4                ; get next count value
   sub.w #$ffff,d4             ; check if end of all modules
   beq done_data               ; if at end, send Restart vector if booting, done otherwise
   move (a2)+,d2               ; get next module count
   bra tx_rx_loop              ; go back to transferring DSP information

done_data:
    bra done_data              ; data file is completed
```

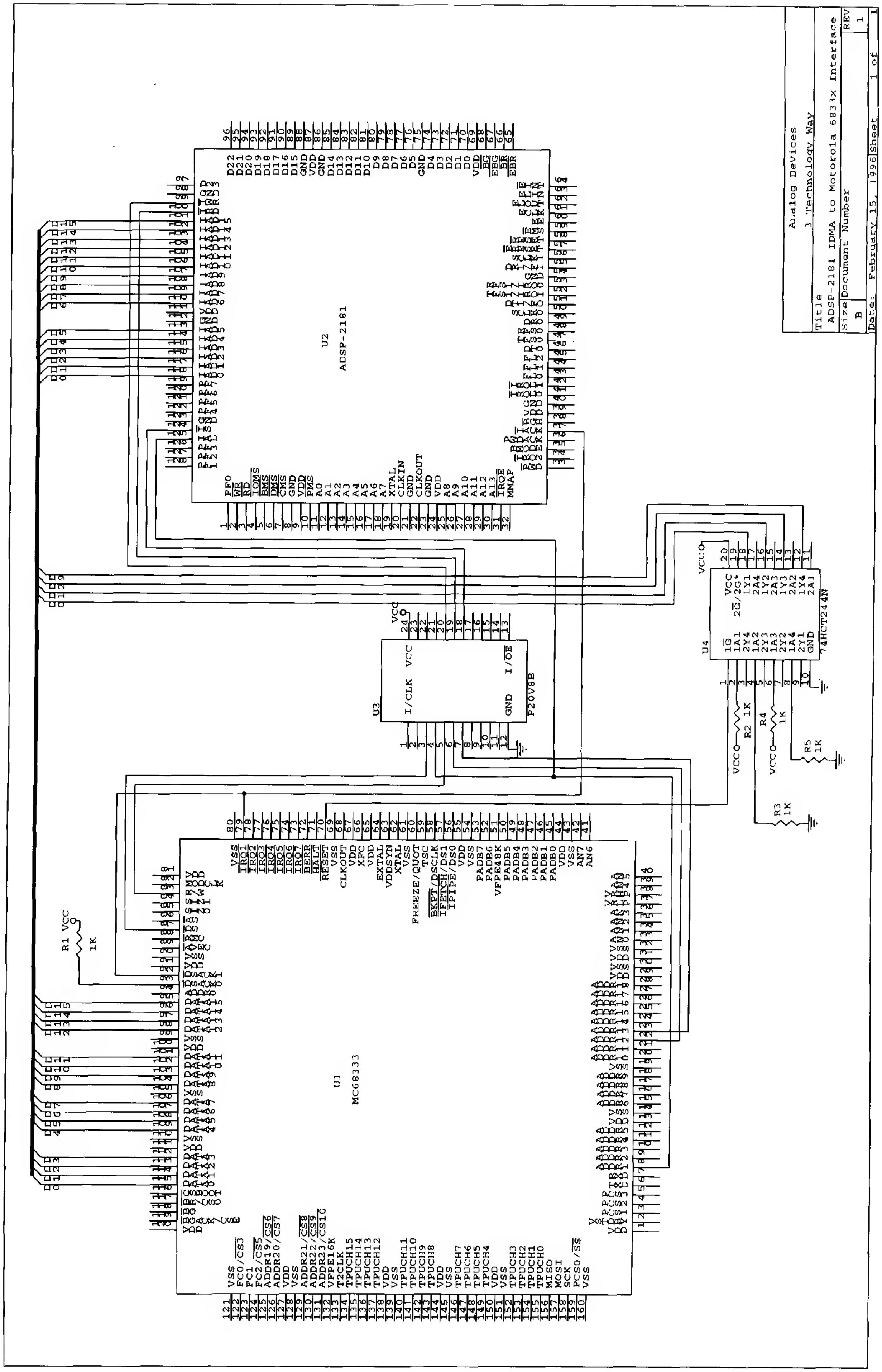
*Motorola MC6833x Assembly Code - download.asm*

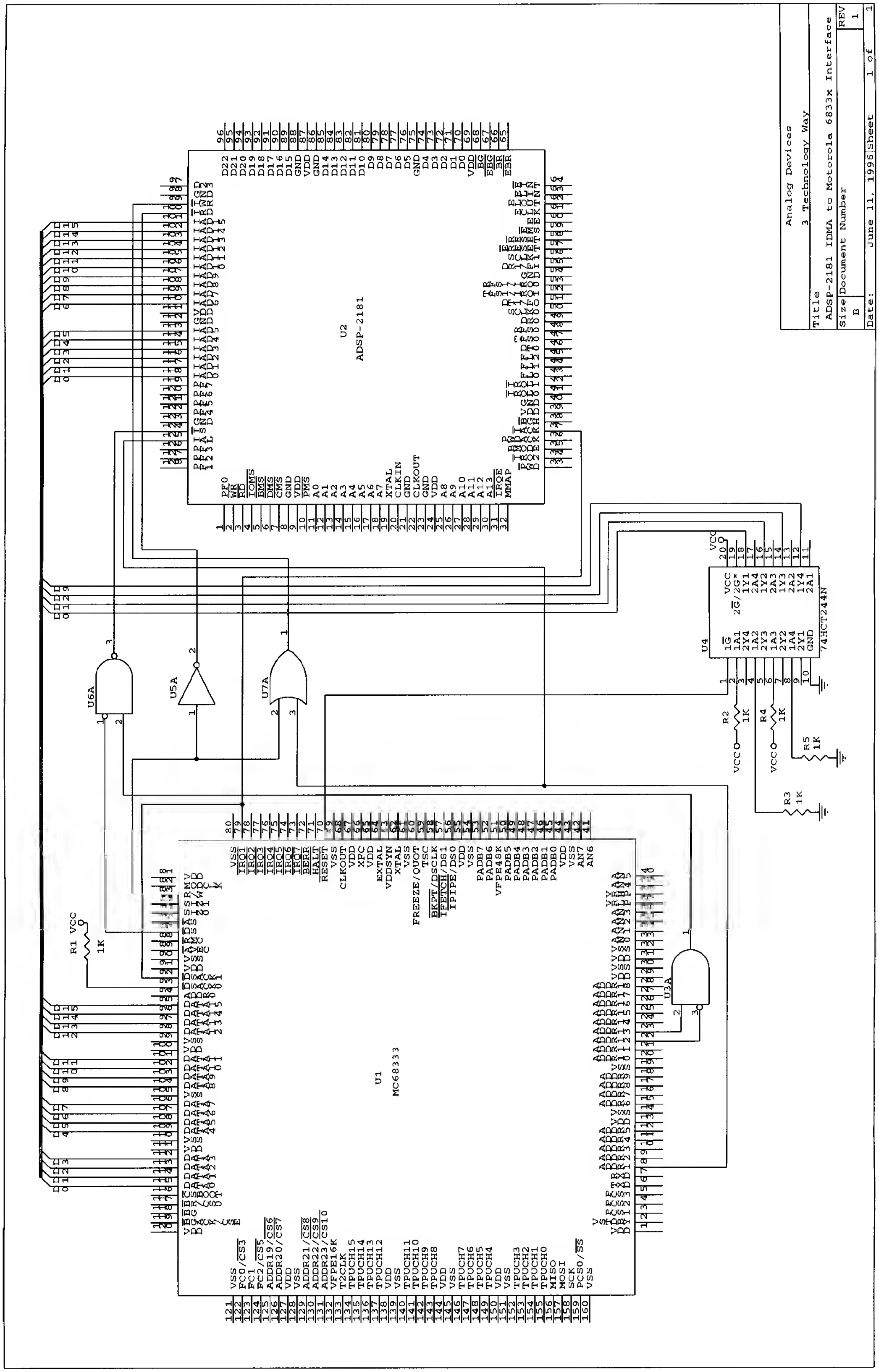
```
/*	idmaboot.c															*/
/*	This program will take an ADSP-2181 executable file and converts it to a data file	*/
/*	readable by the DOWNLOAD.ASM program for the MC6833x. The output of this		*/
/*	file can be used to boot load the ADSP-2181.							*/
/*																	*/
/*	For each memory segment (corresponding to the @PA or @DA identifier) a word		*/
/*	count is performed and that value is placed at the beginning of each section.		*/
/*																	*/
/*	For Program Memory segments in the DSP executable file, the 24-bit opcode is split	*/
/*	across two 16-bit words, MS word first, followed by the LS byte, right justified. For	*/
/*	example, the PM opcode 123456 would be transformed into 1234  0056. The count is	*/
/*	updated accordingly.												*/
/*																	*/
/*	The opcode for the RESET vector (PM(0x0000)) is trapped separately and placed at	*/
/*	the end of the data file.											*/
/*																	*/
/*	This program was tested as a DOS executable (IDMABOOT.EXE), and invoked with		*/
/*	the command line:													*/
/*		type exefile.exe | idmaboot > output.dat							*/
/*																	*/
/*	where exefile.exe is the name of your DSP executable and output.dat is the name of	*/
/*	our desired data file output.										*/
/*																	*/
/*	This program was compiled as a DOS executable using Borland C++, release 4.5		*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define		MAXROWS	1024	/* Declare matrix to store file */
#define		MAXCOLS	16


/* fill_zeros: routine to add leading zeros to PM opcode LS byte */
int		fill_zeros( char *frm, char too[], int fld )
{
	int		i , j ;

		/* fill in left spaces w/ 0's */
		for(i = 0; i <= fld-strlen(frm)-1; i ++) too[i] = '0';

		/* fill in right spaces w/ number */
		for(j = 0; i <= fld-1; i ++, j++) too[i] = *(frm+j);

		/* terminate string w/ NULL */
		too[i] = NULL;

	return ( 1 );
}

main()
{
char		line[MAXROWS][MAXCOLS];
char		first_instr[10][5];
char				hex[12], xxx[12];
int		start, count, i, x, address;

start = 0;
count = 0;
```

```
/* Read in file */
while (fgets((line+count)[0],MAXROWS,stdin) != NULL)
{

	/* Check for start of module */
	if (! start && line[0][0] == '@')
	{
		start = 1;
		count ++;
	}

	/* Check for end of module */
	else if (line[count][0] == '#')
	{
		start = 0;

		/* If PM, write length, address, 2 MSBs, and 1 LSB with leading
	zeros on next line */
		if (line[0][1] == 'P')
		{

			/* Check for RESTART vector. If booting the DSP, PM(0x0000)
			   must be written to last. */
			if (line[1][0] == '0' && line[1][1] == '0' && line [1][2] == '0'
			&& line[1][3] == '0')
			{
				count = count-1;                    /* If RESTART vector,         */
				line[1][3] = '1';                   /* decrement count and        */
				first_instr[0][0] = line[2][0];     /* save opcode for later      */
				first_instr[0][1] = line[2][1];     /* inclusion.                 */
				first_instr[0][2] = line[2][2];
				first_instr[0][3] = line[2][3];
				first_instr[0][4] = line[2][4];
				first_instr[0][5] = line[2][5];
			}
			sprintf ( hex , "%X" , 2*(count-2) );   /* Write count value to file */
			fill_zeros ( hex , xxx, 4);             /* Split opcode and add      */
			printf ( "%s\n" , xxx );                /* leading zeros             */
			printf ( "%s" , (line+1)[0] );
			for (i = 3 ; i <= count-1 ; i ++)
				{
				printf ("%c%c%c%c\n" , (line+i)[0][0], (line+i)[0][1],
				(line+i)[0][2], (line+i)[0][3]);
				printf ("00");
				printf ( "%c%c\n" , (line+i)[0][4], (line+i)[0][5]);
				}

		}

		/* If DM, write length, address (Or'ed with 0x4000), and data */
		else if (line[0][1] == 'D')
```

```
                {

                        sprintf (hex, "%X", 1*(count-2));
                        fill_zeros (hex, xxx, 4);
                        printf ("%s\n", xxx);
                        sscanf((line+1) [0], "%X", &x);
                        address = x | 0x4000;
                        printf ("%X\n" , address);
                        for (i = 2 ; i <= count-1 ; i ++)
                                printf ("%s" , (line+i)[0]);

                }

                count = 0;
        }

        else if (start)
        {
                count ++;
        }

    }

    printf ("0002\n");                                       /* Add RESTART vector to    */
    printf ("0000\n");                                       /* output file. Length = 2  */
    printf ( "%c%c%c%c\n", first_instr[0][0], first_instr[0][1],/* Address = 0000       */
      first_instr[0][2], first_instr[0][3]);
    printf ("00");
    printf ( "%c%c\n", first_instr[0][4], first_instr[0][5]);
    printf ("ffff\n");                                       /* Terminate file with ffff */
}
```

*DSP Code Conversion Program - idmaboot.c*


Analog Devices
3 Technology Way
Title
ADSP-2181 IDMA to Motorola 6833x Interface
Size Document Number REV
B 1
Date: February 15, 1996 Sheet 1 of 1
U1
MC68333
U2
ADSP-2181
U3
P20V8B
U4
74HCT244N
R1 VCC 1K
R2 1K
R3 1K
R4 1K
R5 1K


Analog Devices
3 Technology Way
Title
ADSP-2181 IDMA to Motorola 6833x Interface
Size
B
Document Number
REV
1
Date: June 11, 1996
Sheet 1 of 1
U2
ADSP-2181
U1
MC68333
U4
74HCT244N
U3A
U5A
U6A
U7A
R1 1K
R2 1K
R3 1K
R4 1K
R5 1K

E2125a-X-6/96

PRINTED IN U.S.A.